PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 12/02 — A1

(11) International Publication Number: **WO 95/16958**

(43) International Publication Date: 22 June 1995 (22.06.95)

(21) International Application Number: PCT/US94/14087

(22) International Filing Date: 7 December 1994 (07.12.94)

(30) Priority Data:
08/166,293 13 December 1993 (13.12.93) US

(71) Applicant: CRAY RESEARCH, INC. [US/US]; 655 A Lone Oak Drive, Eagan, MN 55121 (US).

(72) Inventors: PASE, Douglas, M.; 1000 Lavon Lane, Burnsville, MN 55337 (US). WAGNER, Dave; 6560 - 135th Street West, Apple Valley, MN 55124 (US).

(74) Agent: RAASCH, Kevin, W.; Schwegman, Lundberg & Woessner, 3500 IDS Center, 80 South Eighth Street, Minneapolis, MN 55402 (US).

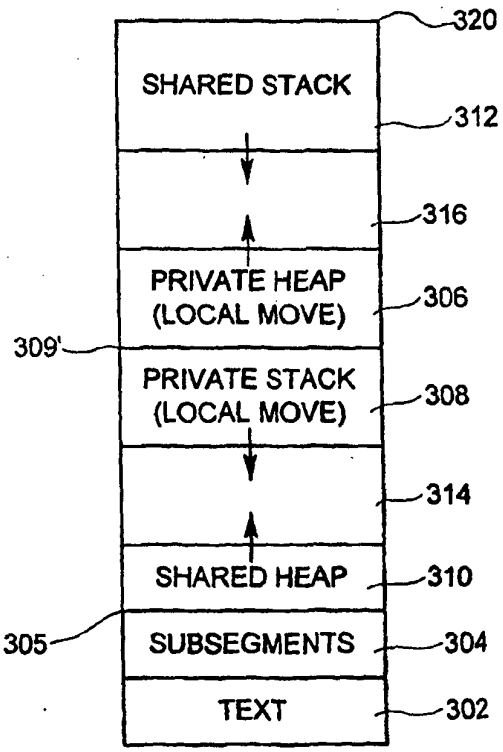
(81) Designated States: CA, JP, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.

(54) Title: MANAGING DISTRIBUTED MEMORY WITHIN A PARALLEL PROCESSING SYSTEM

(57) Abstract

A method of managing distributed memory in which a local memory is partitioned into a shared heap segment, a shared stack segment, a private heap segment and a private stack segment. One of the segment starts at a fixed address and grows upward. A second segment starts at a fixed address and grows downward. A third segment starts at a relocatable segment wall and grows downward and a fouth segment starts at a relocatable segment wall and grows upward.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	**GB**	United Kingdom	**MR**	Mauritania
AU	Australia	**GE**	Georgia	**MW**	Malawi
BB	Barbados	**GN**	Guinea	**NE**	Niger
BE	Belgium	**GR**	Greece	**NL**	Netherlands
BF	Burkina Faso	**HU**	Hungary	**NO**	Norway
BG	Bulgaria	**IE**	Ireland	**NZ**	New Zealand
BJ	Benin	**IT**	Italy	**PL**	Poland
BR	Brazil	**JP**	Japan	**PT**	Portugal
BY	Belarus	**KE**	Kenya	**RO**	Romania
CA	Canada	**KG**	Kyrgystan	**RU**	Russian Federation
CF	Central African Republic	**KP**	Democratic People's Republic of Korea	**SD**	Sudan
CG	Congo			**SE**	Sweden
CH	Switzerland	**KR**	Republic of Korea	**SI**	Slovenia
CI	Côte d'Ivoire	**KZ**	Kazakhstan	**SK**	Slovakia
CM	Cameroon	**LI**	Liechtenstein	**SN**	Senegal
CN	China	**LK**	Sri Lanka	**TD**	Chad
CS	Czechoslovakia	**LU**	Luxembourg	**TG**	Togo
CZ	Czech Republic	**LV**	Latvia	**TJ**	Tajikistan
DE	Germany	**MC**	Monaco	**TT**	Trinidad and Tobago
DK	Denmark	**MD**	Republic of Moldova	**UA**	Ukraine
ES	Spain	**MG**	Madagascar	**US**	United States of America
FI	Finland	**ML**	Mali	**UZ**	Uzbekistan
FR	France	**MN**	Mongolia	**VN**	Viet Nam
GA	Gabon				

MANAGING DISTRIBUTED MEMORY WITHIN A PARALLEL PROCESSING SYSTEM

Field of the Invention

The present invention relates generally to massively parallel processing systems, and more particularly to a memory management protocol used to translate a virtual address on one processing element to a physical address into local memory of a second processing element.

Background of the Invention

Massively parallel processing involves the utilization of hundreds of thousands of processing elements (PE's) linked together by high speed interconnect networks. Typically, each PE includes a processor, local memory and an interface circuit connecting the PE to the interconnect network. A distributed memory massively parallel processing (MPP) system, such as that shown in Fig. 1, is one wherein each processor has a favored low latency, high bandwidth path to one or more local memory banks, and a longer latency, lower bandwidth access over the interconnect network to memory banks associated with other processing elements (remote or global memory). In globally addressed distributed memory systems, all memory is directly addressable by any processor in the system. Typically, to access that global memory, a virtual address generated during program execution must be translated into a physical address into local memory of a processing element.

Even though local memory is addressed globally, it typically remains under the control of its local processor 102. One typical memory management strategy is illustrated in Fig. 3. In Fig. 3, local memory 10 is divided into three parts: text segment 12, heap segment 14 and stack segment 16. Text segment 12 holds program instructions and is placed near the bottom of the local memory address space. Heap segment 14 is used to allocate memory as needed by the execution program. It is placed above text segment 12 and it grows upward. Finally, stack segment 16 resides at the top of the physical memory space and grows downward. Stack segment 16 is

used to store variables in response to an exception or when entering a subroutine and to store data while in the subroutine. Free memory 18 is the unallocated memory between heap segment 14 and stack segment 16.

Heap segment 14 and stack segment 16 tend to grow and shrink as data objects are declared or released within a program. As a program declares new data objects in the main routine, heap segment 14 grows, if needed, to provide memory locations in heap segment 14 needed for the new data object. If, as segment 14 or 16 grows, the amount of free memory 18 drops to zero, a collision occurs between the segments. The user system must then perform some routine such as garbage collection to create more free memory.

The above approach works well for single processing element applications or multiple processing applications where communication between processing elements is limited to message passing. In some situations, it can be advantageous to provide a more integrated view of memory to the processing elements. Such an integrated view can be presented via, for example, a shared memory model of local memory. In such a model, the same areas of memory on different local memories are allocated to the same data object; that data object is distributed across the different local memories. What is needed is an efficient way to manage memory within the shared memory model.

Summary of the Invention

To overcome limitations in the art described above and to overcome other limitations that will become apparent upon reading and understanding the present specification, the present invention provides a method of managing distributed memory in which a local memory is partitioned into a shared heap segment, a shared stack segment, a private heap segment and a private stack segment. One of the segments starts at a fixed address and grows upward. A second segment starts at a fixed address and grows downward. A third segment starts at a relocatable segment wall and grows downward and a fourth segment starts at a relocatable segment wall and grows upward.

Brief Description of the Drawings

The foregoing and other objects, features and advantages of the invention, as well as the presently preferred embodiments thereof, will become apparent upon reading and understanding the following detailed description and accompanying drawings in which:

FIGURE 1 shows a simplified block diagram of a massively parallel processing system in which the memory management protocol can be used;

FIGURE 2 shows a block diagram of a processing element (PE), including a processor, its associated shell circuitry, and local memory;

FIGURE 3 illustrates a message passing local memory structure; and

FIGURES 4A-D illustrates some representative memory segment layouts.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

In the following detailed description of the preferred embodiment, reference is made to the accompanying drawings which form a part hereof, and in which is shown by way of illustration a specific embodiment in which the invention may be practiced. It is to be understood that other embodiments may be utilized and structural changes made without departing from the scope of the present invention.

The preferred MPP system is a MIMD massively parallel processing system with a physically distributed, globally addressable memory. A representative massively parallel processor (MPP) system 100 is shown in FIGURE 1. The MPP system 100 contains hundreds or thousands of processing elements 101, wherein each processing element 101 comprise a processor 102, connected to a local memory 104 and associated support circuitry (not shown). The PE's in the MPP system 100 are linked via an interconnect network 106.

The preferred MPP system 100 has a physically distributed globally addressed memory, wherein each processor 102 has a favored, low latency, high bandwidth path to a local memory 104, and a longer latency lower bandwidth access to the memory banks 104 associated with other processors

102 over the interconnect network 106. A distributed addressing scheme for addressing globally addressed memory in the preferred MPP system 100 is described in "SYSTEM AND METHOD OF ADDRESSING DISTRIBUTED MEMORY WITHIN A MASSIVELY PARALLEL PROCESSING SYSTEM," filed December 10, 1993, by Oberlin et al.

In the preferred embodiment, the interconnect network 106 is comprised of a 3-dimensional torus which, when connected, creates a 3-dimensional matrix of PEs. The torus design has several advantages, including high speed information transfers and the ability to avoid bad communication links. A toroidal interconnect network is also scalable in all three dimensions. An interconnect network of this nature is described in more detail in the copending and commonly assigned U.S. Patent Application Serial Number 07/983,979, entitled "DIRECTION ORDER ROUTING IN MULTIPROCESSING SYSTEMS", by Gregory M. Thorsen, filed November 30, 1992.

FIGURE 2 shows a simplified block diagram of one embodiment of a PE 101. In such an embodiment, an individual PE 101 includes a high-performance RISC (reduced instruction set computer) microprocessor 202. In the preferred MPP system, microprocessor 202 is the DECChip 21064-AA RISC microprocessor, available from Digital Equipment Corporation. Each microprocessor 202 is coupled to a local memory 204 that is a distributed portion of the globally-addressable system memory. Each PE 101 further includes a shell of circuitry that implements synchronization and communication functions facilitating interactions between processors.

The shell circuitry includes an interconnection network router 206, used to connect multiple PEs 101 in a three-dimensional toroidal "fabric." The toroidal interconnect network carries all data communicated between PEs and memories that are not local. In the embodiment shown, an optional block transfer engine 208 in the PE shell circuitry permits asynchronous (i.e., independent of the local processor) movement of data, such as block transfers, between the local memory 204 and remote memories associated with other PEs 200, while providing flexible addressing modes that permit a high degree

of control over the redistribution of data between the shared portions of the system memory. In one such embodiment, a separate hardware-implemented address centrifuge is associated with the block transfer engine 208. However, it shall be understood that the address centrifuge operation can be used to extract PE number and offset from any global address, with or without special hardware support. The implementation of the address centrifuge within the block transfer engine is for purposes of illustration only, and is not a limitation of the present invention.

The shell circuitry also includes a data prefetch queue 210 which allows microprocessor 202 to directly initiate data movement between remote memories and the local processor in a way that can hide access latency and permit multiple remote memory references to be outstanding.

Synchronization circuits in the shell permit synchronization at various different levels of program or data granularity in order to best match the synchronization method that is "natural" for a given parallelization technique. At the finest granularity, data-level synchronization is facilitated by an atomic swap mechanism that permits the locking of data on an element-by-element basis. The atomic swap operation is described in detail in the copending and commonly assigned U.S. patent application entitled "ATOMIC UPDATE OF MEMORY," filed on October 22, 1993, by Barriuso et al. A more coarse grain data-level synchronization primitive is provided by a messaging facility, which permits a PE to send a packet of data to another PE and cause an interrupt upon message arrival, providing for the management of message queues and low-level messaging protocol in hardware. Control-level synchronization at the program loop level is provided by a large set of globally accessible fetch-and-increment registers that can be used to dynamically distribute work (in the form of iterations of a loop, for instance) among processors at run time. Yet another form of control-level synchronization, barrier synchronization, is useful to control transitions between major program blocks (i.e., between loops performing actions on the same data sets). The barrier mechanism is described in detail in the copending and commonly assigned U.S. patent application entitled "BARRIER

SYNCHRONIZATION FOR DISTRIBUTED MEMORY MASSIVELY PARALLEL PROCESSING SYSTEMS," filed December 10, 1993 by Oberlin et al.

Memory Management

In one embodiment of MPP system 100, a parallel FORTRAN is implemented with a shared memory model. Such a model has been found to be useful in reducing the burden on programmers; it is much easier to write programs using shared memory than to use message passing. In the preferred MPP system 100, both shared and private data objects are supported. Such an approach tends to give the user the greatest flexibility in exploiting locality. Shared objects allow easy cooperation across processing elements while retaining some locality. Private objects allow the programmer to maximize locality at the processing element at some cost to ease of use.

In the preferred embodiment of MPP system 100, a processor 102 attempting to read a remote memory location provides a physical address into the local memory 104 on the other processing element. That is, there is no translation of virtual memory bits to physical memory locations on the remote processing element. This approach results in a system in which a processing element 101 can reference memory in a remote local memory 104 without interrupting the processor 102 associated with that local memory 104. In such embodiment, however, all shared memory must be maintained at the same predetermine locations in the local memories 104 of the processing elements 101 assigned to a particular task in order that it can be accessed by the processing elements 101 assigned to that task.

In order to maintain both shared and private data objects, in one embodiment, local memory 104 is split into at least four segments: a shared heap, a shared stack, a private heap and a private stack in local memory. In one embodiment, the segments are placed such that one end of each segment is set at a fixed address. Such an embodiment can be understood in the context of Fig. 4a. In Fig. 4a, local memory 300 comprises a text segment 302, a subsegments segment 304, a private heap segment 306, a private stack segment 308, a shared heap segment 310 and a shared stack segment 312.

Private heap segment 306 is placed at a fixed address 305 above subsegment segment 304 and grows upward from fixed address 305. Private stack segment 308 is placed at a fixed address 309 and grows downward. Shared heap segment 310 is placed at fixed address 310 and grows upward. Finally, shared stack segment 312 is placed at the top of local memory and grows downward. Free memory 314 is the amount of unused memory between segments 306 and 308 while free memory 316 is the amount of unused memory between segments 310 and 312. Operating system software executing on the processor 102 associated with the remote local memory keeps track of the extent of each segment.

In one embodiment of local memory 300, each segment 306, 308, 310 and 312 is fixed at one end of the segment. This means that the amount of free memory available to each segment is limited to that free memory 314 or 316 allocated to the area between segments 306 and 308 and between 310 and 312. For example, in the embodiment of Fig. 4a described above, free memory 314 can be used only by either private heap 306 or private stack segment 308. In such an embodiment, when private heap 306 and private stack 308 grow into each other, there is no recourse. None of free memory 316 can be used. This can result in premature exhaustion of local memory.

In an alternate embodiment of local memory 300 in Fig. 4a, address 309 can be relocated under control of operating system software executing on processor 102 of the processing element. In such an embodiment, if free memory between two segments runs out because the two segments have grown into each other (a segment collision), the operating system relocates address 309 and moves the contents of segments 308 and 310 accordingly. This move frees up more memory between the colliding segments and execution can therefore continue. In one embodiment, processor 102 reads the contents of the memory locations to be moved, stores the contents in a register in processor 102 and then writes the contents to the relocated memory locations.

This alternate approach to local memory 300 has the advantage that both free memory 314 and 316 must run out of allocable memory locations

before a collision occurs which could be terminal to the program. This approach does have one drawback. On a collision, when the shared heap is moved to begin at a new physical address, processing elements which might want to access that shared heap must wait until all shared memory has been allocated on a processing element before they access that processing element's local memory. (It is generally not safe to permit access to a shared memory location on a processing element until you know that the processing element has allocated the memory.) In addition, processing elements must synchronize even if the shared segment move is due solely to an increase in the size of one of the private segments. In one embodiment, each processing element waits for a barrier synchronization signal before it accesses another processing element's shared memory space.

Another approach to allocating shared and private memory is illustrated in Fig. 4b. In Fig. 4b, shared segments 310 and 312 have been placed at the ends of local memory 320 and private segments 306 and 308 have been grouped into the middle of local memory 320. As above, fixed address 309 can be relocated under control of operating system software executing on processor 102 of the processing element. In such an embodiment, if free memory between two segments runs out because the two segments have grown into each other (a segment collision), the operating system relocates address 309 and moves the contents of segments 306 and 308 accordingly. This move frees up more memory between the colliding segments and execution can therefore continue. In one embodiment, during the segment move, processor 102 reads the contents of the memory locations to be moved, stores the contents in a register in processor 102 and then writes the contents to the relocated memory locations.

The approach of Fig. 4b is advantageous in that it eliminates the synchronization needed when shared memory is moved due to a private memory collision. In Fig. 4b, the memory moved is strictly private segments and, therefore, under the complete control of the processor 102 on that processing element. Synchronization may still be necessary in order to

coordinate the growing or shrinking of a shared segment, even if there is not segment collision as a result.

It should be apparent that one does not have to keep private segments 306 and 308 contiguous in order to profit from the approach of Fig. 4b. In Fig. 4c, a free memory area 332 in local memory 330 could be used to create some buffering between private heap segment 306 and private stack segment 308. Free memory area 332 can be used to limit the amount of moving one private segment when a collision occurs between its neighboring private segment and a shared segment. In one embodiment, a certain amount of free memory is kept in free memory 332. In such an embodiment, an understanding of a program's behavior could be used to select an optimum free memory area 332 in order to reduce the number of times that two segments must be moved. In one embodiment, a certain amount of free memory is kept in free memory 332. As the segments begin to occupy more of the available memory, free memory 332 is reduced until it is zero. In such an embodiment, behavior will tend toward the behavior of memory 320 as free memory 332 goes to zero.

Finally, it is conceivable that one might wish to combine message passing with the shared model in a multiprocessor application. Such an embodiment is shown in Fig. 4d. In local memory 340 in Fig. 4d, private heap 342 and private stack 344 are used to store messages passed from one processing element 101 to the next. Therefore, each move of the private heap 342 and private stack 344 must be synchronized with the processing elements in order to avoid the situation where one processing element writes into another's local memory before the collision move is completed. Such synchronization can be ensured through the use of the barrier synchronization mechanism described in "BARRIER SYNCHRONIZATION FOR DISTRIBUTED MEMORY MASSIVELY PARALLEL PROCESSING SYSTEMS," filed December 10, 1993, by Oberlin et al.

It is clear that the memory management techniques described above provide a flexible approach to segment management which minimizes the complexity of address translation by keeping shared memory segments at

known locations in memory. This application is intended to cover any adaptations or variations of the present invention and is therefore limited only by the claims or equivalents thereof.

What is claimed is:

1. A method of allocating memory within a local memory of a processing element in a computer system having a plurality of processing elements connected by an interconnect network, the method comprising the steps of:

allocating a first segment which begins at a first fixed address within local memory and grows upward;

allocating a second segment which begins at a second fixed address within local memory and grows downward;

defining a segment wall within local memory, wherein the segment wall is relocatable under control of the operating system;

allocating a third segment which begins on a first side of the segment wall and grows downward; and

allocating a fourth segment which begins on a second side of the segment wall and grows upward.

2. The method according to claim 1 wherein the first segment is a shared heap and the second segment is a shared stack.

3. The method according to claim 2 wherein the third segment is a private stack and the fourth segment is a private heap.

4. A computer system comprising:

a plurality of processing elements, wherein each processing element comprises a processor and local memory connected to the processor; and

an interconnection network connecting the processing elements;

wherein the local memory of each of the processing elements is configured to include:

a first segment which begins at a fixed address within local memory and grows upward;

a second segment which begins at a fixed address within local memory and grows downward;

first and second segment walls within local memory, wherein the segment walls are relocatable under control of the operating system;

a third segment which begins on one side of the first segment wall; and

a fourth segment which begins on one side of the second segment wall;

wherein the local memory segment wall is relocatable under control of the operating system.

5. The system according to claim 4 wherein the first and second segment walls are located at a particular memory location.

6. A method of allocating memory within a local memory of a processing element in a computer system having a plurality of processing elements connected by an interconnect network, the method comprising the steps of:

allocating a first segment which begins at a first fixed address within local memory and grows upward;

allocating a second segment which begins at a second fixed address within local memory and grows downward;

defining first and second segment walls within local memory, wherein the segment walls are relocatable under control of the operating system;

allocating a third segment which begins on one side of the first segment wall and grows downward; and

allocating a fourth segment which begins on one side of the second segment wall and grows upward.

7. The method according to claim 6 wherein the first segment is a shared heap and the second segment is a shared stack.

8. The method according to claim 6 wherein the third segment is a private stack and the fourth segment is a private heap.

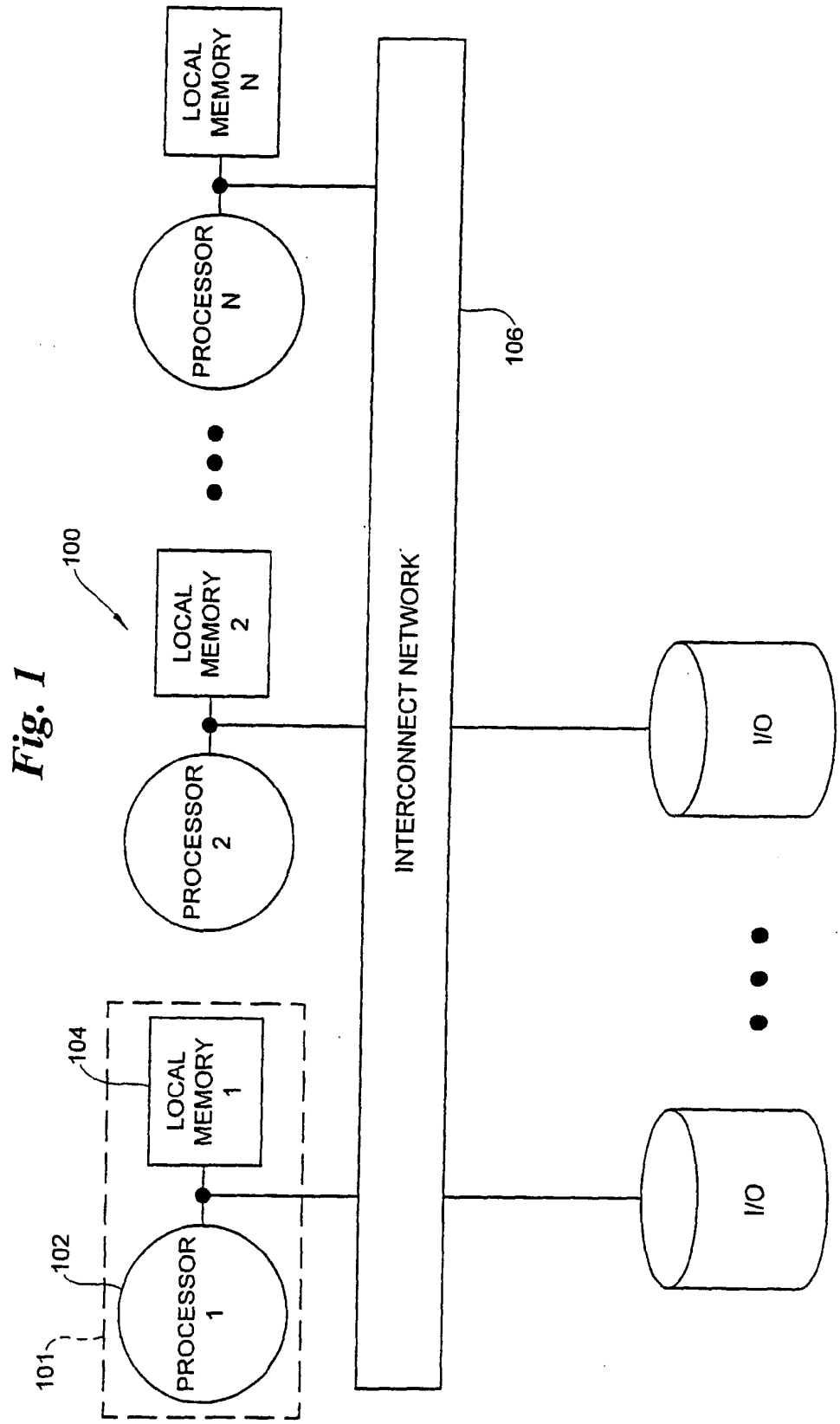

100
101
102
104
106
PROCESSOR 1
LOCAL MEMORY 1
PROCESSOR 2
LOCAL MEMORY 2
PROCESSOR N
LOCAL MEMORY N
INTERCONNECT NETWORK
I/O
I/O

Fig. 1

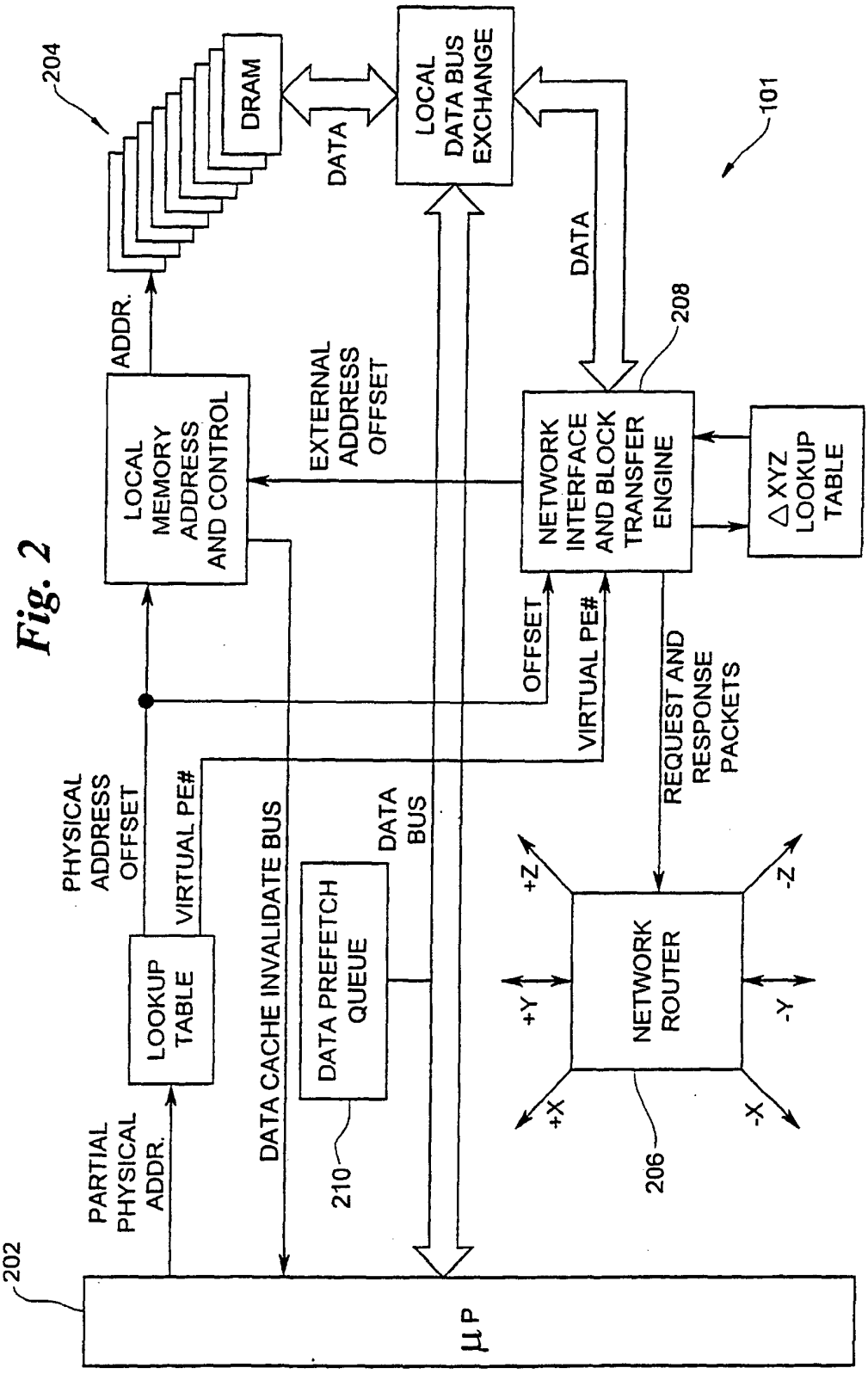


Fig. 2

Fig. 3

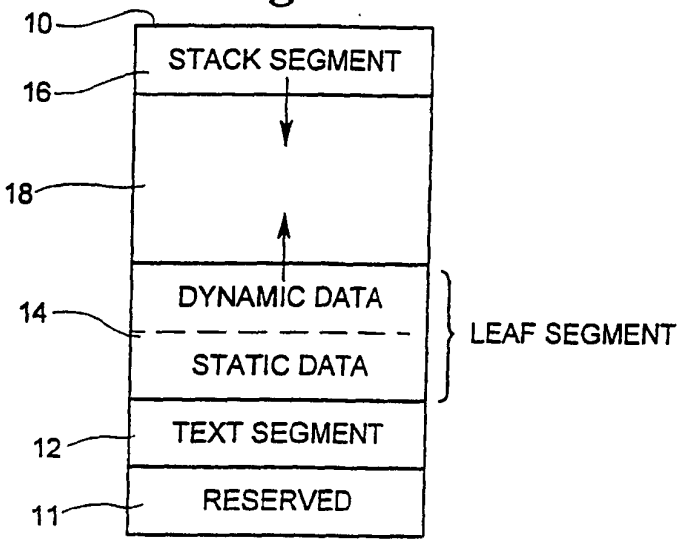


Fig. 4b

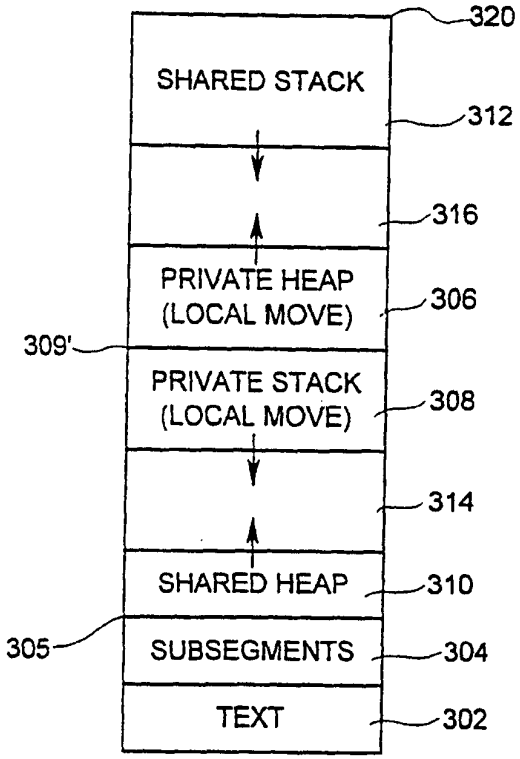


Fig. 4a

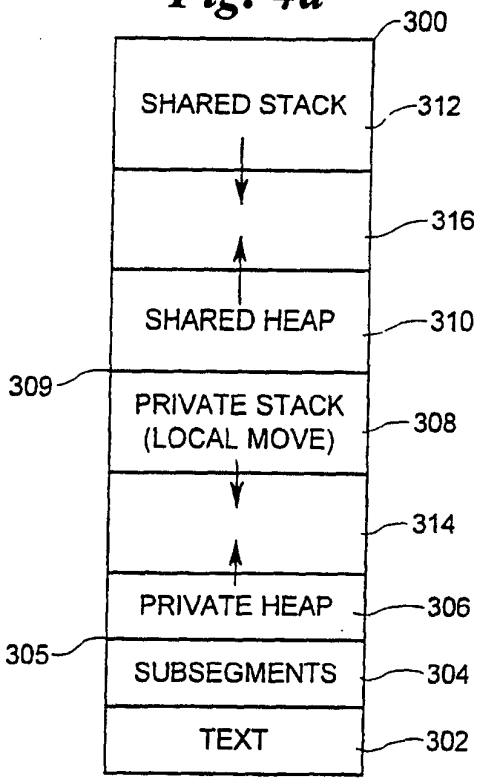


Fig. 4c

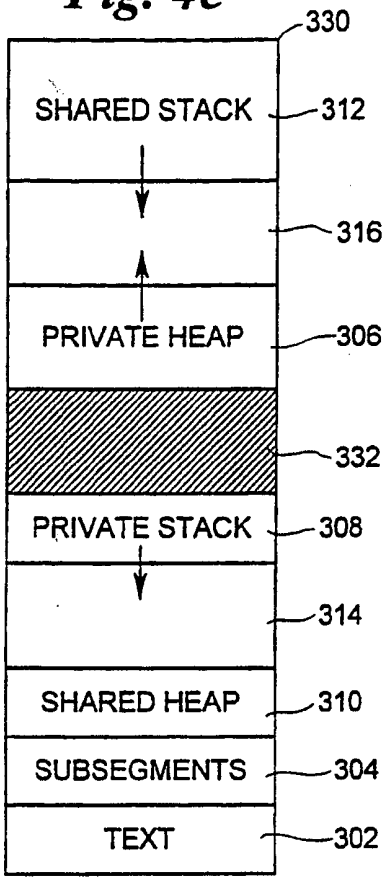
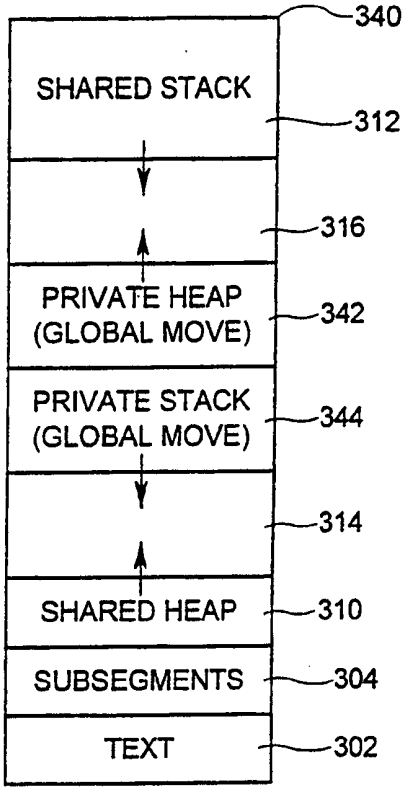

330
SHARED STACK
312
316
PRIVATE HEAP
306
332
PRIVATE STACK
308
314
SHARED HEAP
310
SUBSEGMENTS
304
TEXT
302

Fig. 4d


340
SHARED STACK
312
316
PRIVATE HEAP
(GLOBAL MOVE)
342
PRIVATE STACK
(GLOBAL MOVE)
344
314
SHARED HEAP
310
SUBSEGMENTS
304
TEXT
302

INTERNATIONAL SEARCH REPORT

Inter nal Application No
PCT/US 94/14087

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F12/02

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US,A,3 924 245 (EATON ET AL) 2 December 1975	1,4,6
A	see column 6, line 42 - line 53 see column 4, line 7 - line 43; figure 2	2,3,5,7, 8
A	EP,A,0 572 696 (IBM) 8 December 1993 see page 3, line 22 - page 4, line 26; figure 2	1,4,6

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
27 February 1995	13. 03. 95
Name and mailing address of the ISA European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax (+31-70) 340-3016	Authorized officer Ledrut, P

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

Inter ...nal Application No
PCT/US 94/14087

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US-A-3924245	02-12-75	GB-A- 1441816	07-07-76
		DE-A,B,C 2431379	06-02-75
		FR-A,B 2238188	14-02-75
		JP-C- 1072324	30-11-81
		JP-A- 50043844	19-04-75
		JP-B- 56012902	25-03-81
EP-A-0572696	08-12-93	JP-A- 6103157	15-04-94